Grant Agreement: 287829

Comprehensive Modelling for Advanced Systems of Systems

COMPASS

Second release of the COMPASS Tool User Manual

Technical Note Number: D31.2a

Version: 1.2

Date: January 2013

Public Document

http://www.compass-research.eu

Contributors:

Joey W, Coleman, AU
Anders Kaels Malmos, AU
Rasmus Lauritsen, AU
Luis D. Couto, AU

Editors:

Joey Coleman, AU

Reviewers:

Ralph Hains, Atego
Simon Foster, York

Document History

Ver	Date	Author	Description
0.1	03-01-2013	LDC	Initial document version
0.2	08-01-2013	RL	Type Checker section added
—	—	LDC	POG and Simulation sections added
—	—	AKM	Simulation section expanded
—	—	JWC	Draft Introduction and Conclusion added
1.0	09-01-2012	JWC	Ready for internal review but screenshots need to be updated
—	09-01-2012	RWL	Added section on how to reference directories in the file system
1.1	23-01-2012	JWC	Editing based on internal review
1.2	28-01-2012	JWC	(More) Editing based on internal review

Contents

1 Introduction **5**

2 Obtaining the Software **6**

3 The Command-line Interface **7**
3.1 Available Functionality 7
3.2 Basic Invocation 8
3.3 CML Simulation 9

4 Using the COMPASS Perspective **12**
4.1 Eclipse Terminology 12

5 Managing COMPASS Projects **15**
5.1 Creating new COMPASS projects 15
5.2 Importing COMPASS projects 15
5.3 Referencing folders in the filesystem 17
5.4 Exporting COMPASS projects 18

6 The CML Type Checker **21**
6.1 Output 21
6.2 Representation 21

7 Using the Proof Obligation Generator **23**

8 The COMPASS Simulator **24**
8.1 Using COMPASS Simulator 24
8.2 Creating a Launch Configuration 25
8.3 Launch Via Shortcut 27
8.4 Simulation 28

9 Conclusion **31**

1 Introduction

This document is a user manual for the COMPASS tool, an open source tool supporting systematic engineering of System of Systems using the COMPASS Modelling Language (CML). The ultimate target is a tool that is built on top of the Eclipse platform, that integrates with the RT-Tester tool and also integrates with Artisan Studio. For developers and advanced users, the core CML language functionality is also available from a command-line interface. This document is targeted at users with limited experience working with Eclipse-based tools. Directions are given as to where to obtain the software.

This user manual does not provide details regarding the underlying CML formalism. Thus if you are not familiar with this, we suggest the tutorial for CML before proceeding with this user manual [WCF+12, BGW12]. However, users broadly familiar with CML may find the Tool Grammar reference (COMPASS Deliverable D31.2c [Col13]) useful to ensure that the they are using the exact syntax accepted by the tool.

This version of the document supports version 0.1.0 of the COMPASS tool suite. The intent is to introduce readers to how this version of the tool interacts with CML models. The connection to the Artisan Studio tool is not yet available and, hence, is not described further in this deliverable.

The main tool is the COMPASS IDE, which integrates all of the available CML analysis functionality and provides editing abilities. It also integrates the CML simulator and initial version of the link to the RT-Tester tool.

We also provide a command-line interface to the core functionality of the tool. This interface is aimed primarily at developers but there are situations where it may be useful to users of the tool.

Section 2 describes how to obtain the software and install it on your own computer. Section 3 describes the command-line interface to the COMPASS tool. Section 4 explains the different views in the COMPASS Eclipse perspective. This is followed by Section 5 which explains how to manage different projects in the COMPASS tool. Section 6 describes what output the CML typechecker will produce, and where it may be found in the COMPASS IDE. Section 7 describes how to access the output from the proof obligation generator. Section 8 describes the interface to the COMPASS simulator as included in the COMPASS IDE.

2 Obtaining the Software

This section explains how to obtain the COMPASS IDE and COMPASS command-line tool, described in this user manual.

The COMPASS suite is an open source tool, developed by universities and industrial partners involved in the COMPASS EU-FP7 project [FLW12]. The tool is developed on top of the Eclipse platform.[1]

The source code and pre-built releases for the COMPASS CML tool are hosted on SourceForge.net, as this has been selected as our primary mechanism for supporting the community of users of CML and the developers building tools for the COMPASS platform. It has facilities for file distribution, source code hosting, and bug reporting.

The simplest way to run the COMPASS Tool is to download it from the SourceForge.net project files download page at

`https://sourceforge.net/projects/compassresearch/files/`

This download is a specially-built version of the Eclipse platform that only includes the components that are neccessary to run the COMPASS Tool — it does not include the Java development tools usually associated with the Eclipse platform.

Once the tool has been downloaded, in order to run it, simply unzip the archive into the directory of your choice and run the COMPASS executable. The tool is self-contained so no further instalation is necessary.

Also available from that page is the command-line tool, which exposes the core language analysis functionality via a command-line interface. This tool is intended for developers and advanced users.

The COMPASS CML tool requires the Java SE Runtime Environment version 6 or later. On Windows environments, either the 32-bit or 64-bit versions may be used, on Mac OS X and Linux, the 64-bit version is required.

Artisan Studio and the RT-Tester environment are available from Atego and Verified Systems International, respectively, and are not distributed through the SourceForge.net website. Obtaining those software environments is outside of the scope of this document.

[1] `http://www.eclipse.org`

3 The Command-line Interface

The command-line interface to the COMPASS tool was conceived as a tool for developers to quickly allow them to access and test the core libraries. This allows developers of the tool to quickly test new functionality for correctness without having to create the GUI elements that will control the functionality in the integrated IDE. A beneficial side-effect of having this tool is that general users are not required to load the IDE to test CML programs, but instead may invoke them via the command-line.

3.1 Available Functionality

The command-line tool presently has access to the following features in the core libraries:

- CML parser
- CML typechecker
- CML AST to DOT graph generation
- CML proof obligation generator
- CML interpreter
- Example core plugins

The CML parser is the primary element of the command-line tool, as nothing can happen without using it. Generally, the tool will read in a (sequence of) CML file(s) and then perform a typecheck on the abstract syntax tree (AST). At this point, the data is ready to be used by the rest of the core libraries and plugins. It is possible to run the core libraries on an AST that has not been typechecked, but doing so is not recommended except to test error reporting or if the user only wishes to generate a DOT graph of the AST.

The DOT graph generator will output a representation of the AST generated from the input CML files in the DOT language. The output is suitable for use in the Graphviz suite of graph visualization utilities.[2] The output is useful for producing a visual representation of the data used internally by the COMPASS tool to represent the static structure of a model of a system of systems. This allows a developer to quickly verify whether the input CML files result in the expected internal data structures.

[2]Found at `http://www.graphviz.org`.

The Proof Obligation Generator (POG) can be invoked by the command-line tool and doing so will cause it to produce the internal representation of the consistency and validation checks that the input CML files require. The theorem proving and model checking plugins planned for future releases will be able to use these proof obligations to verify the consistency and correctness of the input CML model.

The CML interpreter is only accessible from the command-line tool in the M12 release of the COMPASS tools. Invoking the CML interpreter on a set of input CML files will result in the model being executed in a simulation run. The results of the simulation will be printed to the console during the run. Interfaces to graphical components are not yet available.

3.2 Basic Invocation

After obtaining the commandline tool package, decompress it into a folder. In that folder will be –among others– the files `cmlc` and `cmlc.bat`. Invocation of the `cmlc` (Linux, Mac OS X) or `cmlc.bat` (Windows) script with no parameters will produce the following output:

```
COMPASS command line CML Checker - CML M16

Usage: cmlc [switches] <file1>, ...,<fileN>
Switches:
    -coe    -   Continue on Exception
    -dotg   -   DOT graph generation,
                -dotg=<out> write output to <out>
    -dwa    -   Run the Div Warn Analysis example
    -e      -   Simulation,
                -e=<processId> simulate the process identified
                   by <processId>
    -empty  -   Empty analysis, run the empty analysis
    -i      -   Interactive mode
    -notc   -   Omit type checking phase
    -po     -   Parse Only, stop analysis after the parsing
    -soe    -   Silence on Exception
    -tco    -   Type Check Only
```

Assuming some CML model in a file, `example.cml`, loading it into the command-line interface is accomplished by typing `cmlc example.cml`. If run in this manner, the output will be:

```
COMPASS command line CML Checker - CML M16
Parsing file: example.cml
1 file(s) successfully parsed. Starting analysis:
 Running The CML Type Checker on example.cml
```

Note that, by default, the interpreter is not invoked on input; see Section 3.3.

It is also possible to input CML directly into the command-line tool when invoked with the `-i` option. This is useful for quickly cutting and pasting small bits of CML, for example.

To generate a DOT-language graph representation of a parsed CML model, we use the `-dotg=<file>` option. The invocation `cmlc -dotg=example.gv example.cml` will produce console output:

```
COMPASS command line CML Checker - CML M16
Parsing file: example.cml
1 file(s) successfully parsed. Starting analysis:
 Running eu.compassresearch.ast.preview.DotGraphVisitor on
    example.cml
 Running The CML Type Checker on example.cml
```

And it will also write out the file `example.gv` in the process. This file can then be processed with a DOT language processor (such as Graphviz) into many other formats, including PDF, SVG, PNG, and JPEG.

3.3 CML Simulation

The commandline tool enables simulation of CML models when invoked with the `-e` option. Since the CML model may have more than one process defined, the `-e=<processId>` option must be supplied, where `<processId>` is the name of the process that is to be simulated.

As an example of how this works, consider the following CML model in a file called `example.cml`:

```
channels
  init, a, b

process A = begin
  @ init -> a -> Skip
end
```

```
process B = begin
  @ init -> b -> Skip
end

process C = A;B
```

The following command will simulate the process identified by `C`:

```
cmlc -e=C example.cml
```

This results in the following output being printed to the console:

```
COMPASS command line CML Checker - CML M16
Parsing file: example.cml
1 file(s) successfully parsed. Starting analysis:
 Running The CML Type Checker on example.cml
 Running The CML Interpreter on example.cml

--------begin step---------
Offered Events:
<init>
Current interpretation state:
C = (A = (init->a->Skip);B)
Trace after step:
<init>

--------begin step---------
Offered Events:
<a>
Current interpretation state:
C = (A = (a->Skip);B)
Trace after step:
<init><a>

--------begin step---------
Offered Events:
<init>
Current interpretation state:
C = (B = (init->b->Skip))
Trace after step:
<init><a><init>

--------begin step---------
Offered Events:
<b>
```

```
Current interpretation state:
C = (B = (b->Skip))
Trace after step:
<init><a><init><b>

--------begin step---------
Offered Events:
Current interpretation state:
Trace after step:
<init><a><init><b>
```

The output has three pieces of information for each step:

Offered Events: These are the events that were available for synchronisation before the current step is taken. This means that the collection of processes in the model are able to synchronise with the environment on these events.

Current interpretation state: This is a representation of the current process state of the interpreter just before the step was taken.

Trace after step: This is the history of past events, including the event that happened in this step. Presently this is the entire history and will produce long outputs.

At present the interpreter will synchronise on any offered event using a stub "environment". This means that there is, in effect, a process running in parallel with every model that can synchronise on every possible event. This is not meant to mirror the intended semantics, but is just an intermediate state that will lead to an interactive mode that allows the user to act as the environment for the purposes of simulating the whole CML model of a system of systems.

4 Using the COMPASS Perspective

When the COMPASS tool is started, the splash screen from Figure 1 should appear. The first time it is started you will have to decide where you want the default place for your projects to be. Click *ok* to start using the default workspace and close the *welcome* screen to get started for the first time.

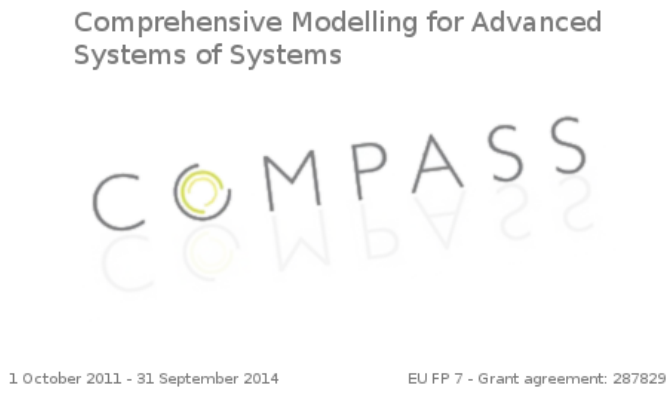
Comprehensive Modelling for Advanced
Systems of Systems
COMPASS
1 October 2011 - 31 September 2014
EU FP 7 - Grant agreement: 287829

Figure 1: The COMPASS spash screen used at startup

4.1 Eclipse Terminology

Eclipse is an open source platform based around a *workbench* that provides a common look and feel to a large collection of extension products. Thus, for a user familiar with one Eclipse product it will generally be easy to start using a different product on the same workbench. The Eclipse workbench consists of several panels known as *views*, such as the COMPASS Explorer view at the top left of Figure 2. A collection of panels is called a *perspective*, for example Figure 2 shows the standard COMPASS perspective. This consists of a set of views for managing COMPASS projects and viewing and editing files in a project. Different perspectives are available in COMPASS as will be described later, but for the moment think of a perspective as a useful composition of views for conducting a particular task.

The *COMPASS Explorer view* lets you create, select, and delete COMPASS projects and navigate between the files in these projects, as well as adding new files to existing projects.

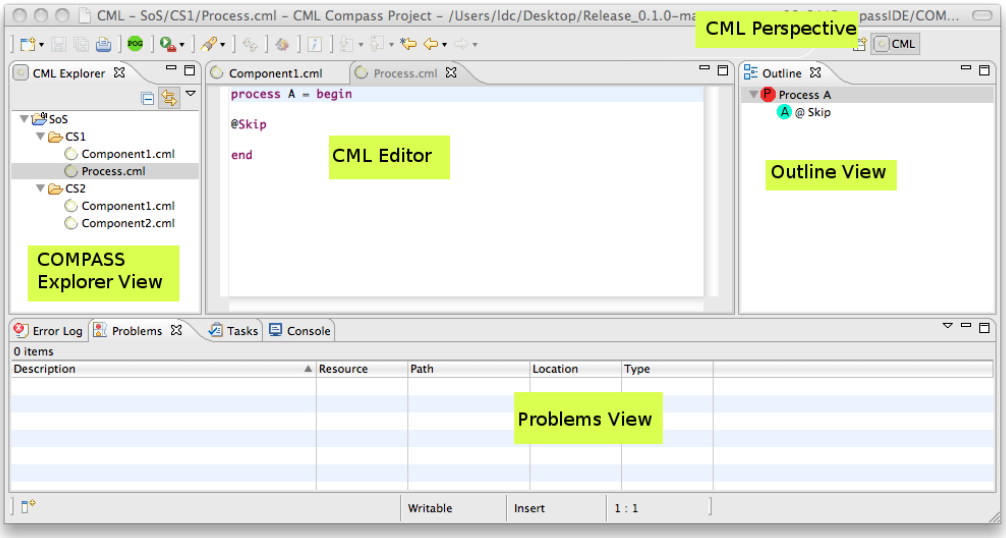

Figure 2: Outline of the COMPASS Workbench.

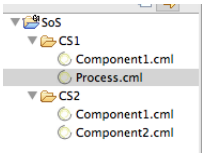

Figure 3: The COMPASS Explorer view.

The *Outline view*, on the right hand side of Figure 2, presents an outline of the file selected in the editor. This view displays any declared CML definitions such as their state components, values, types, functions, operations and processes[3]. The type of the definitions are also shown in the *outline view*. The *Outline view* is at the moment only available for the *CML* models of the system. In the case another type of file is selected, the message *An outline is not available* will be displayed.

The outline will have an appropriate structure based on the type of CML construct found in the source file that is displayed in the visible CML editor. In Figure 4 a CML class is outlined on the left reflecting the structure of a class. On the right Figure 4 depicts a CML process and lists its actions. In the current version of the COMPASS tool outline decorations are omitted but are planned to be as follows: The icon in front of a name indicates the type of respective CML element: a brown circle with a “V” indicates a value, a purple circle with a “T” indicates a Type, a red circle with a “P” indicates a process, a blue circle with an “O” indicates an

[3]In a later version of the tool the outline view will also support other types of files.

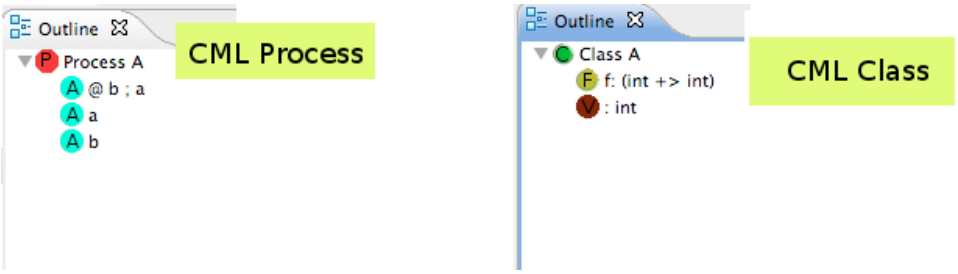

Figure 4: The outline view showing *CML* class named Component1 on the left. On the right the outline view is showing a *CML* process and its actions.

operation, a yellow circle with a "F" indicates a function, a green circle with a "C" indicates a class, a dark brown circle with "Cs" indicates a chanset and a light brown circle with "Ch" indicates a channel.

The higher level elements of the outline begin collapsed and can be expanded to show their child nodes. For example, a process can be expanded in order to see its actions, operations etc.

Clicking on the name of a definition will move the cursor in the editor to the definition. The outline will also automatically highlight whichever node corresponds to the cursor position as it changes.

The outline is only created/refreshed when the source file is saved.

5 Managing COMPASS Projects

This section explains how to use the tool to manage COMPASS projects. Step by step instructions for importing, exporting and creating projects will be given.

5.1 Creating new COMPASS projects

Follow these steps in order to create a new COMPASS project:

1. Create a new project by choosing *File* → *New* → *Project* → *COMPASS project* (see Figure 5)
2. Type in a project name (see Figure 6)
3. Click the button *Finish*.

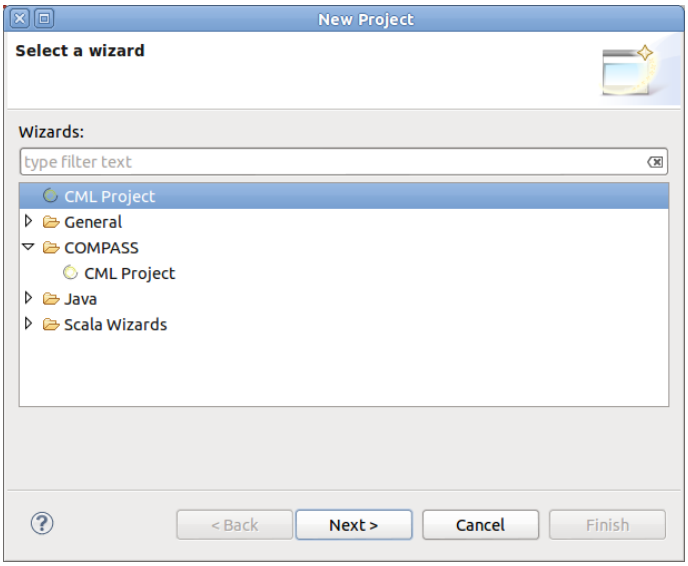

Figure 5: Create Project Wizard

5.2 Importing COMPASS projects

Follow these steps in order to import an already existing COMPASS project:

1. Right-click the explorer view and select *Import*, followed by *General* → *Existing Projects into Workspace*. See Figure 7 for more details. Click *Next* to proceed.

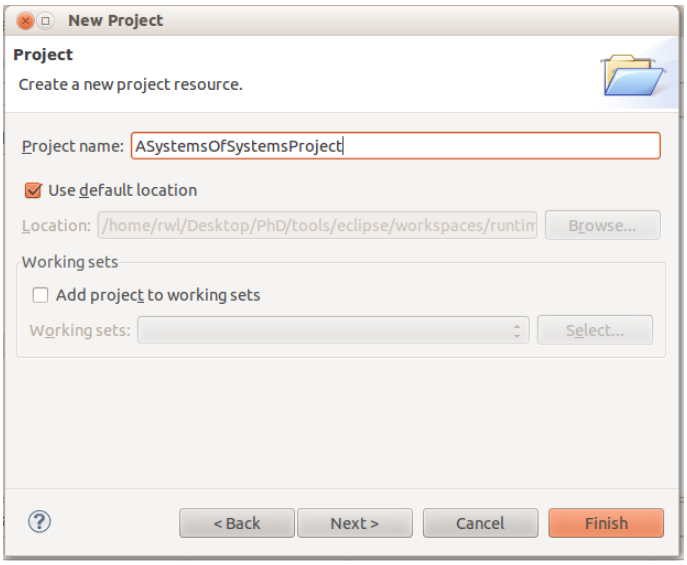

Figure 6: Create Project Wizard

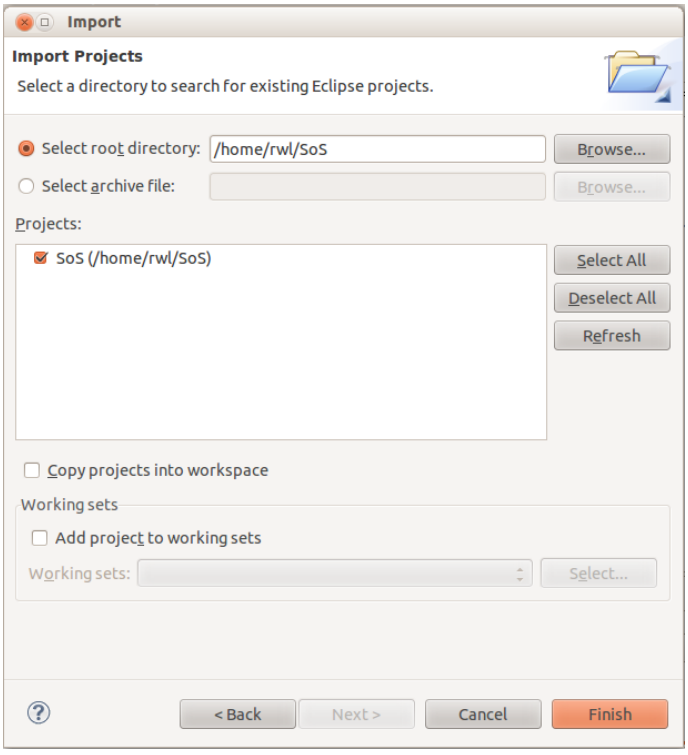

Figure 7: Import Project Wizard

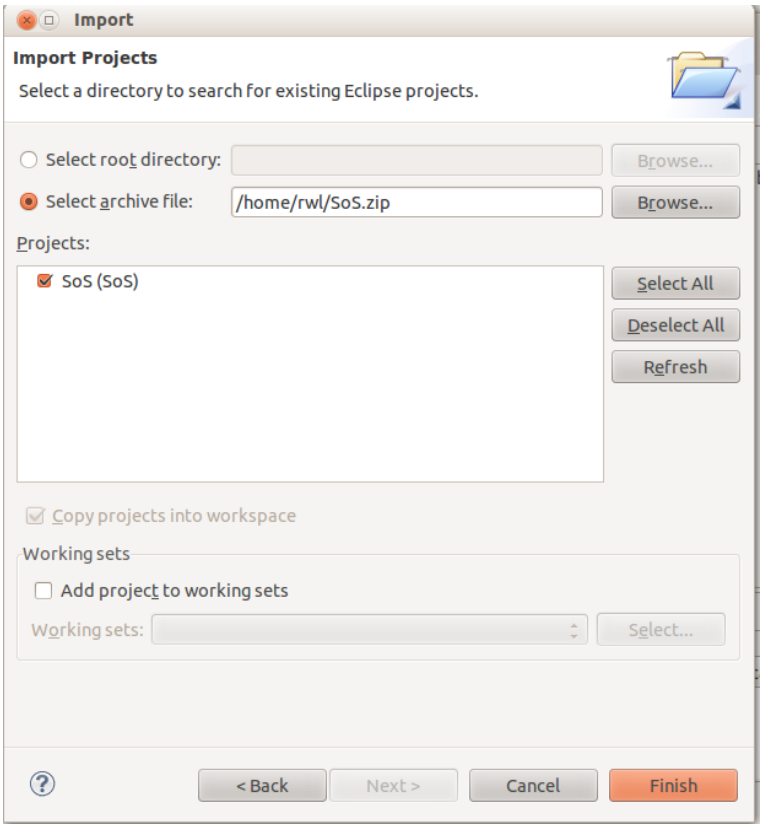

Figure 8: Import archive file

2. If the project is contained in a folder, select the radio button *Select root directory*, if it is contained in a compressed file select *Select archive file*. See Figure 8 for more details.
3. Click on the active *Browse* button and navigate in the file system until the project to be imported is located.
4. Click the button *Finish*. The imported project will appear on the *COMPASS explorer view*.

5.3 Referencing folders in the filesystem

Importing CML files into an Eclipse project means copying the files from the filesystem into the Eclipse workspace. However for CML-files in the Case Study examples it is typically preferrable to edit the files in the file system directly. In this way committing changes to the COMPASS subversion repository will not

include copying changed files from the Eclipse workspace back to the COMPASS subversion check out directory.

To do this it is recommended to create a project in which a link to an external folder can be created. The steps to set this up in the COMPASS Tool are as follows:

1 Create a CML project and give it a name

2 Right click on the project and select New → Other ...

3 In the appearing window, expand General and select Folder, press Next

4 Give the "link" a name of your linking in "Folder name"-field.

5 Press the Advanded button

6 Select the bullet point saying "Link to Alternate location (Linked Folder).

7 Click the "Browse..."-button

8 Navigate to the CML source folder. E.g. compass_svn\ Common\ CaseStudies\ EmergencyResponse\ workspace\ Expert-Led\ model

9 Press "Open"

10 Press "Finish"

Now a new folder with a small link ornament on it will show up in the newly create project. Expand that folder in the "COMPASS Explorer" to see your files.

The Parser and TypeChecker are currently not run at all times even though Automatic Build is enabled. To force the source files to be built go to "project → clean ...".

5.4 Exporting COMPASS projects

Follow these steps in order to export a COMPASS project:

1. Right click on the target project and select *Export*, followed by *General* → *Archive File*. See Figure 9 for more details.
2. A new window like the one shown in Figure 10 will follow. In this case the selected project will appear as root node on the left side of it. It is possible to browse through the contents of the project and select the convenient files to be exported. All the files contained in the project will be selected by default.

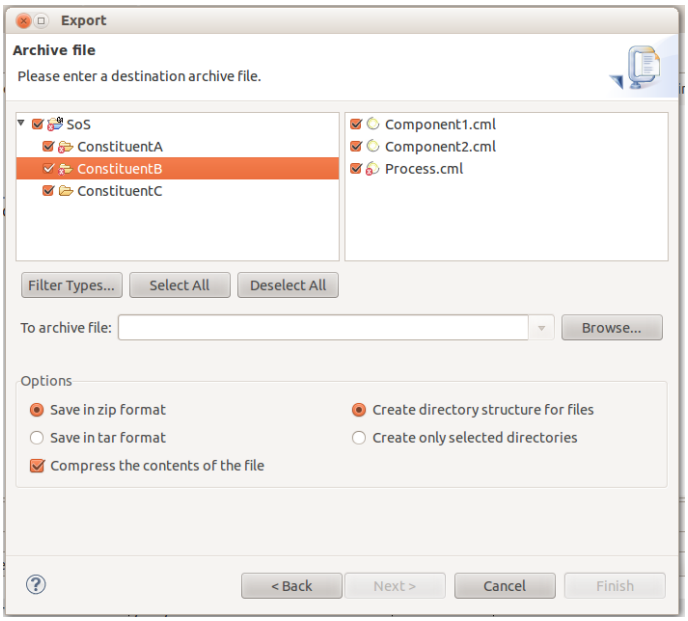

Figure 9: Select an output format for the exporting process.

3. Enter a name for the archive file in the text box following *To archive file*. A specific path to place the final file can be selected through the button *Browse*.

4. Click on the *Finish* button to complete the export process.

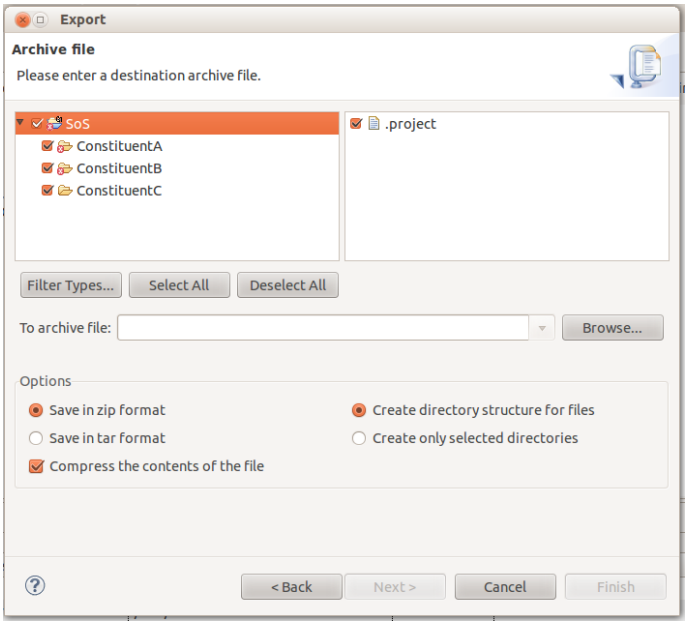

Figure 10: Project ready to be exported.

6 The CML Type Checker

The COMPASS-Tool ships with the CML Type Checker. The Type Checker checks type consistentcy and referential integrity of your model. Type consistentcy includes checking that operator and variable types are respected. Referential integrity includes checking that named references exists and have an appropriate type for their context.

6.1 Output

The type checker produces two kinds of artifacts: Type Errors and Type Warnings. Both carry a reference to the offending bit of the model, a description of what is ill formed and an exact location of where the issue occurred.

6.2 Representation

In the COMPASS-Tool user interface type errors show up in three places. To point the user at the exact piece of CML-source causing an error, an error marker will be showing in the left margin of its Editor. Additionally, the offending piece of syntax will be underlined with red as seen in Figure 11. To give the complete picture for

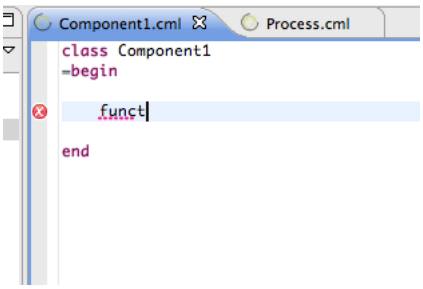

Figure 11: User Interface showing a typical type error marker.

all errors in a given model the problem view shows the list of all generated errors (see Figure 12). The third and last way type errors are made visible in the user interface is through the CML Project Explorer. The CML Project Explorer offers a tree view of CML model file structure. If an error occurs in a CML-source file then all of folders containing that file up through the hierarchy to the project level will have a red error marker (see Figure 13). Type error markers will be updated whenever a CML-Source file is saved with changes. To force a re-check of all

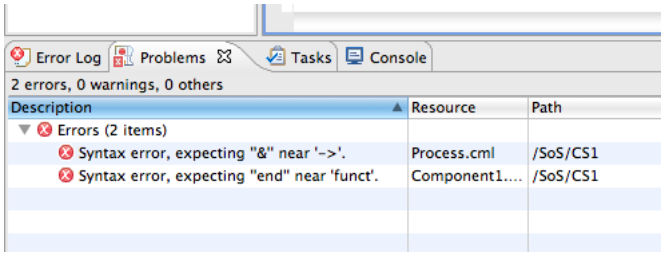

Figure 12: User Interface problems view with type errors.

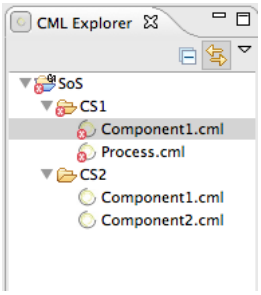

Figure 13: CML Project Explorer with error markers.

source files again click the “Clean ...” option in the Project menu bar, see Figure 14. One last thing to notice is that the Outline view, when displayed, is only

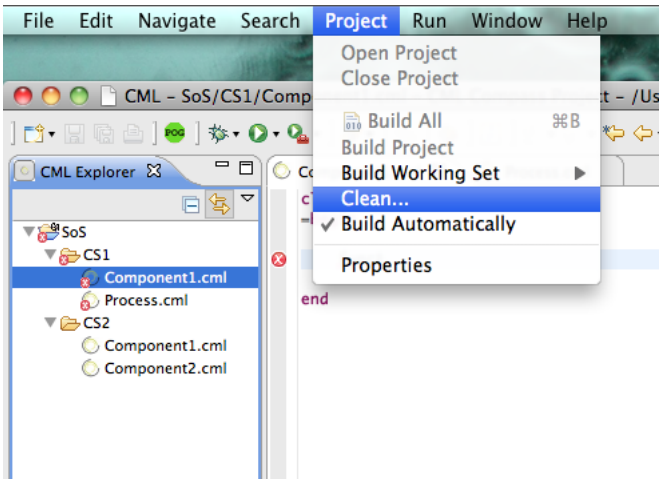

Figure 14: Project menu-bar with “Clean...” option for rechecking the model.

updated for source-files that parse correctly. Thus, files that have parse errors will not have their Outline view updated and may also contain type errors. Seeing an outline is only an indication the model is syntactically correct.

7 Using the Proof Obligation Generator

Usage of the COMPASS Proof Obligation Generator (POG) is quite simple. At the moment, the POG has only one function: generating the POs. In order to do this, the user must click the POG button as shown in Figure 15.

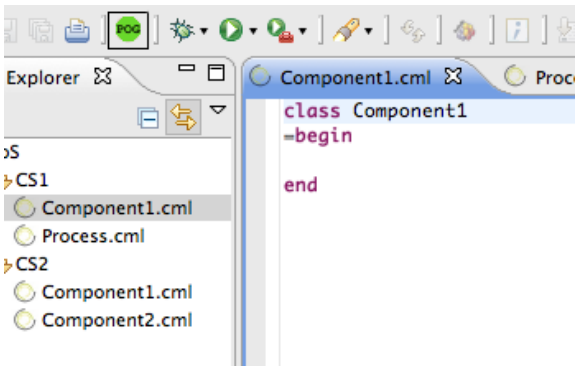

Figure 15: Using the COMPASS POG.

Once the POG has run successfully, the generated POs are written to a file as shown in Figure 16. At the moment the only things that may be done with this file is to inspect and copy (parts of) it. If you wish to discharge any PO, you must do so manually. Keep in mind that the file is temporary and will be deleted when you exit the COMPASS tool. If you wish to preserve the POs, copy the contents of the file elsewhere.

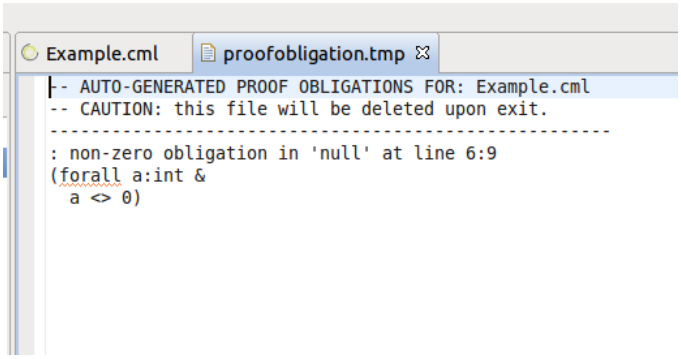
Example.cml
proofobligation.tmp
-- AUTO-GENERATED PROOF OBLIGATIONS FOR: Example.cml
-- CAUTION: this file will be deleted upon exit.
: non-zero obligation in 'null' at line 6:9
(forall a:int &
a <> 0)

Figure 16: The POG output file.

8 The COMPASS Simulator

This section explains how to configure, start, run and stop a *simulation* of a CML model with the COMPASS tool. This involves describing how a simulation launch configuration can be configured and how the simulation engine is launched and used.

There are two ways of simulating a CML model:

1. **Without user interaction**: This option will simulate the model without user interaction. When faced with an observable event choice this will be resolved by picking random one. The events are picked in a random but deterministic manner. Thus, the simulation will always make the same choices for every run of the same model.
2. **With user interaction**: This option will simulate the model without any interaction from the user so long as no observable events occur. If an observable event occurs at some point, then the user must to choose an event before the simulation can continue. Presently the simulator requires a "choice" from the user even when the set of choices is a single event; while this is useful for understanding the behaviour of the model, later versions will allow single-choice events to be automatically chosen.

The first option is associated with the Eclipse "Run" command and the second with the "Debug" command.

As the simulation without user interaction chooses synchronisation events in a pseudo-random manner, we have implemented this in a deterministic manner to allow for easy replay of the exact same sequence of event synchronisations. Future versions of the simulator will enhance the ability to control the sequence of event synchronisation further.

8.1 Using COMPASS Simulator

In order to use the simulator, the CML perpective must be edited in order to make the relevant commands visible. Go to `Window` → `Customize Perpective` → `Command Groups Availability` and enable the `Launch` group (see Figure 17).

Before starting a simulation, a launch configuration must be created. The purpose of this is to define which CML process should be the initial point of simulation. There are two ways of doing this:

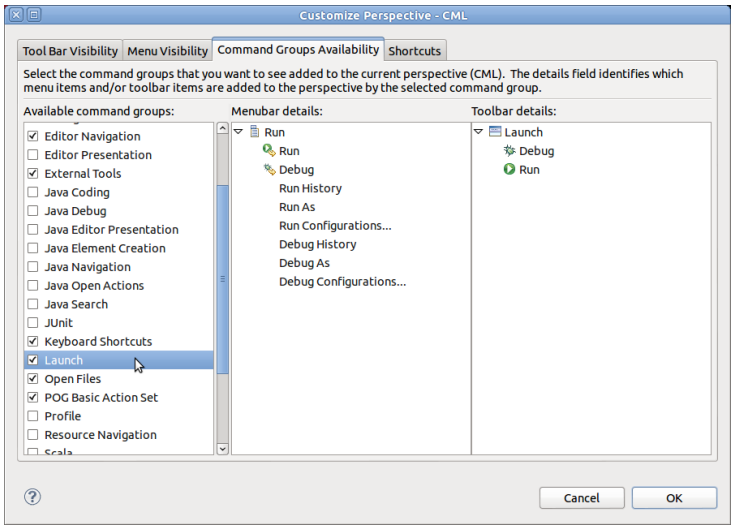

Figure 17: Enabling the Launch Command Group in the CML pespective.

1. Manually create and edit a launch configuration
2. Launch via a shortcut and it will be automatically created

Both will we be explained in greater detail below. The difference between them is the number of steps you have to take before a launch is commenced. So the end result of the two options can be identical, but the first option gives greater configuration possibilities than the second one.

8.2 Creating a Launch Configuration

To create a launch configuration you first click on the small arrow next to either the debug button or the run button as shown in Figure 18.

Once clicked, a drop-down menu will appear with either *Debug configurations* or *Run configurations* (depending on which button you clicked); selected the appropriate *configurations* option. This will open a configurations dialog like the one shown in Figure 19. All of the existing CML launch configurations will appear under "CML Model". To create a new one double-click on "CML Model" then an empty launch configuration will appear on the left as shown in Figure 19. To edit an existing one, click on the desired launch configuration name and the details will appear on the left.

As seen in Figure 19 a project name and a process name needs to be associated with a launch configuration. When choosing a project, you can either write the

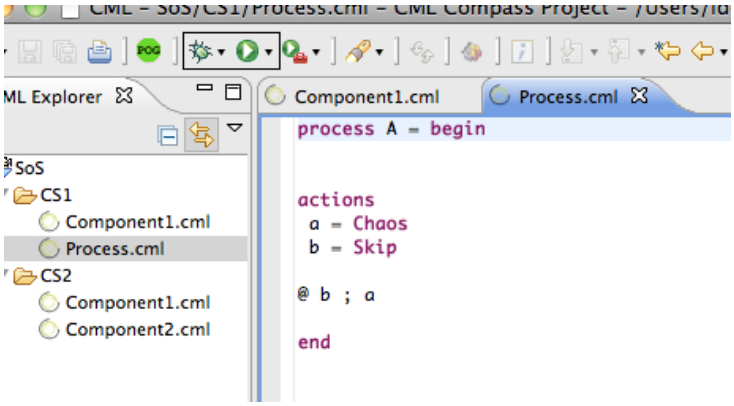

Figure 18: The debug button (the left one) and the run button (the right one), present in the toolbar at the COMPASS tool.

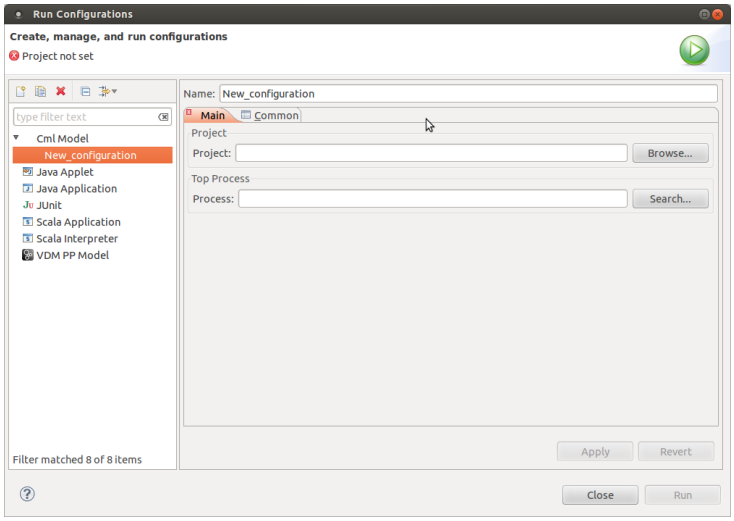

Figure 19: The launch configuration dialog showing a newly created launch configuration

name or click on the *browse* button which shows a list of all the available projects and choose one from there. The process name selection is identical.

The project name and process name must exist. It will not be possible to launch if they do not. In the left corner of Figure 19 a small red icon with an "X" and a message will indicate what is wrong. In the figure it indicates that no project has been set so this should be the first thing to do.

After setting the project name and process name, the *Apply* button must be clicked to save the changes to the launch configuration. If the project exists and is open

and a process with the specified name exists in the project, then the *Run* or *Debug* button will be active and it is possible to launch the simulation as shown in Figure 20.

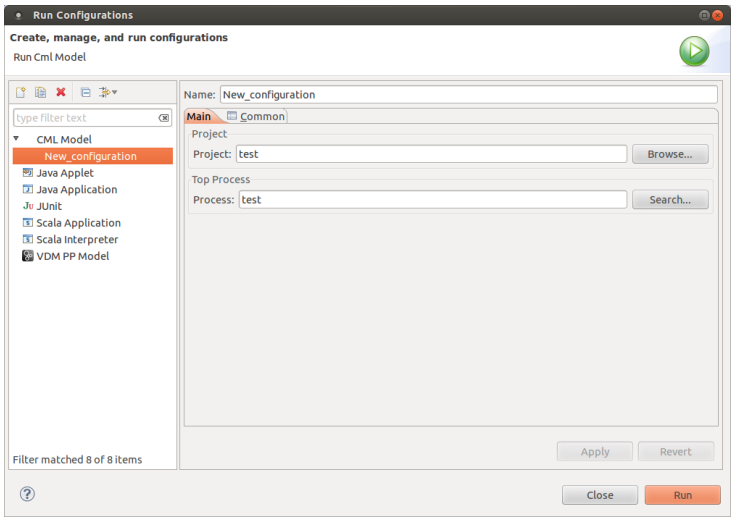

Figure 20: The *configuration dialog* after a project and process has been selected

This launch configuration will now appear in the a drop-down menu described in the beginning of this section. The actual simulation will be described in Section 8.4.

8.3 Launch Via Shortcut

Another way to launch a simulation is through a shortcut in the COMPASS explorer view in the CML perspective. To access this, right click on a cml file to make the context menu appear. From here either choose "Debug As → Debug CML Model" or "Run As → Run CML Model". After that two things can happen: if the cml file only contains one process then this process will be launched, if however more than one process is defined then a process selection dialog appears with a list of possible processes. This is shown in Figure 21.

To launch a simulation, a process must be chosen. This is done by double-clicking one of the process names in the list. This will launch a simulation with that process as the top-level process.

If you launch via a shortcut then a temporary launch configuration will be created and launched. It will not be shown in the list as it would be if it was created as defined in section 8.2.

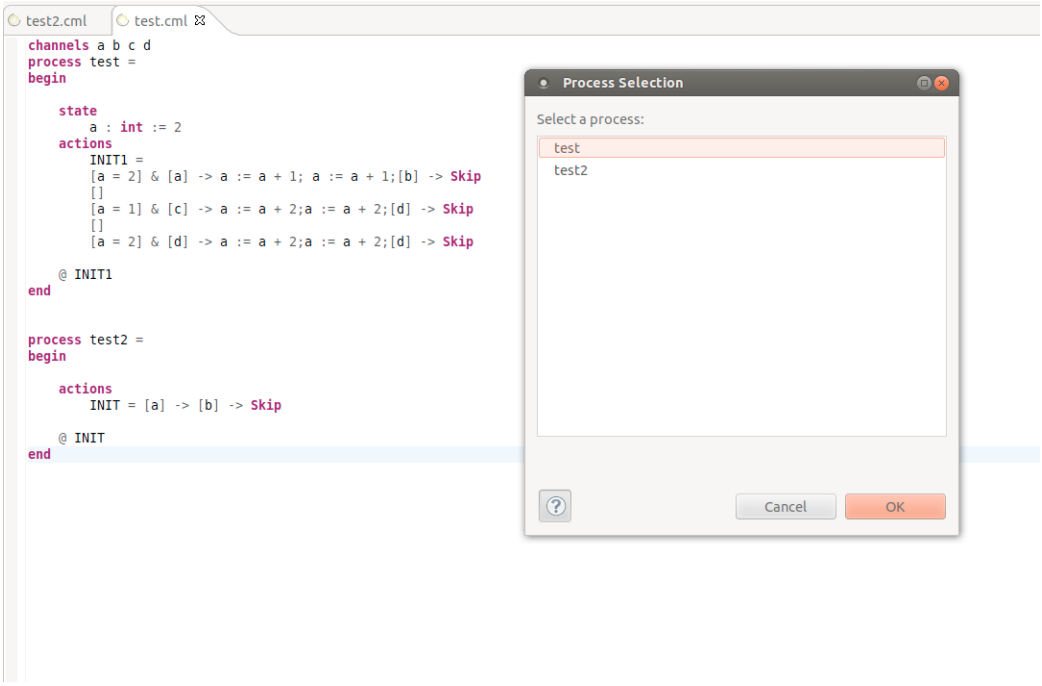

Figure 21: Right after "Run As → Run CML Model" has been clicked on, the context menu of the test.cml file appears. Since the file has defined more than one processes, the *process selection dialog* is shown.

8.4 Simulation

As mentioned in Section 8 there are two possible ways to simulate a model, each of them will be described.

8.4.1 Simulating without user interaction

Simulating without user interaction is achieved by choosing the "Run" option. Once a "Run" simulation is launched, as described in the previous sections, the output of the interpreter is entirely shown in the Eclipse Console view. This is shown in Figure 22.

8.4.2 Simulating with user interaction

Simulating with user interaction is achieved by choosing the "Debug" option. The output of the interpreter is still shown in the console view but there is an additional

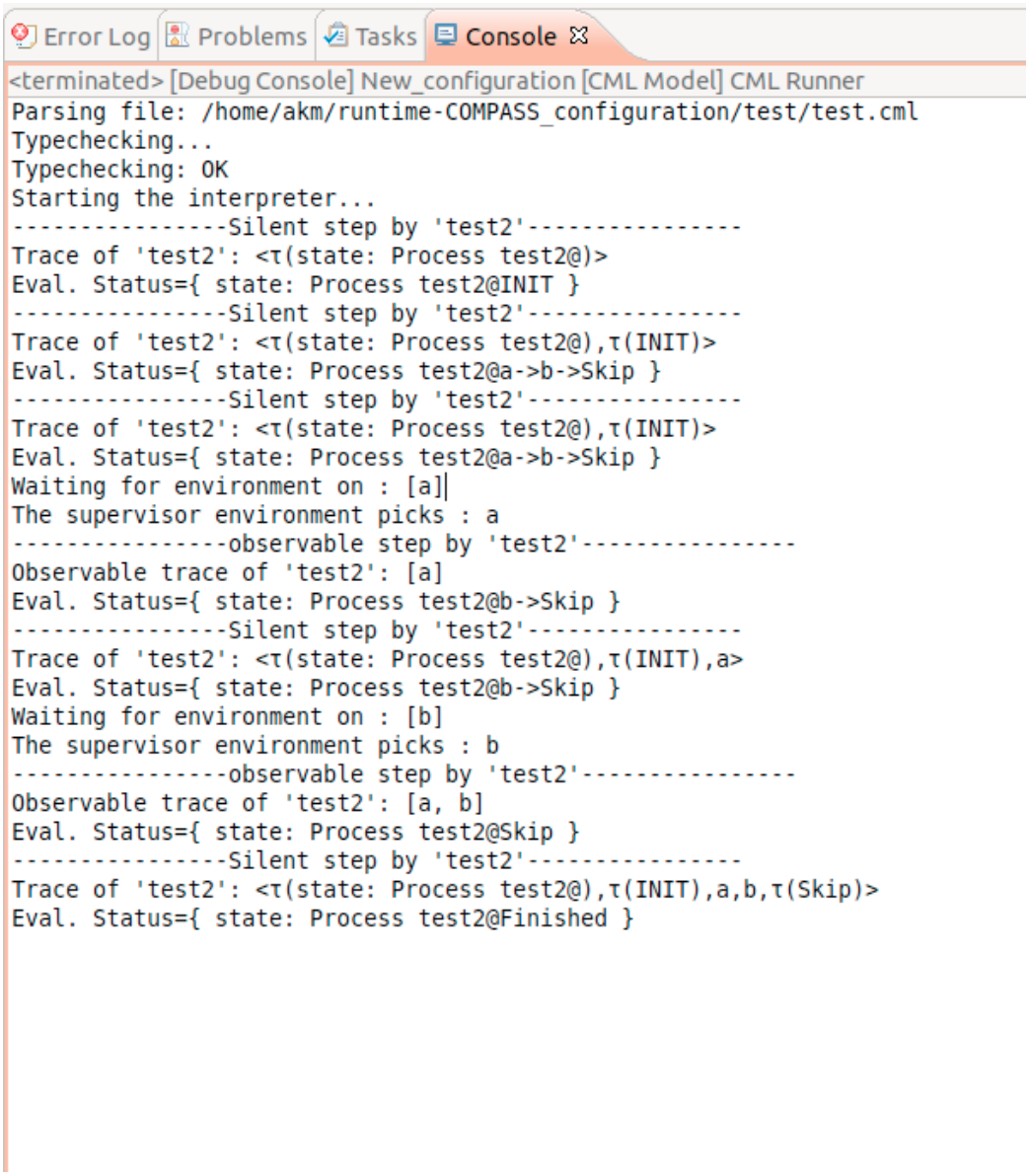

Figure 22: The process “test2” has just been simulated without user interaction

Eclipse View to interact with the simulation, the “CML Event Options” View. This is located in the bottom (in the same location as the console and problems view) if it is shown. If it is not shown, then it can be shown in the following way: go to “Window → Show View → Other...” now the “Show View” dialog appears now select “COMPASS → CML Options View”. Since the output gets focus every times something happens, the “CML Event Options” View should be moved to the left besides the outline. At the moment, this must be done manually

by dragging it with the mouse.

When a debug simulation is launched the perspective changes to the Eclipse Debug Perspective. Additional information will appear here. This is shown in Figure 23

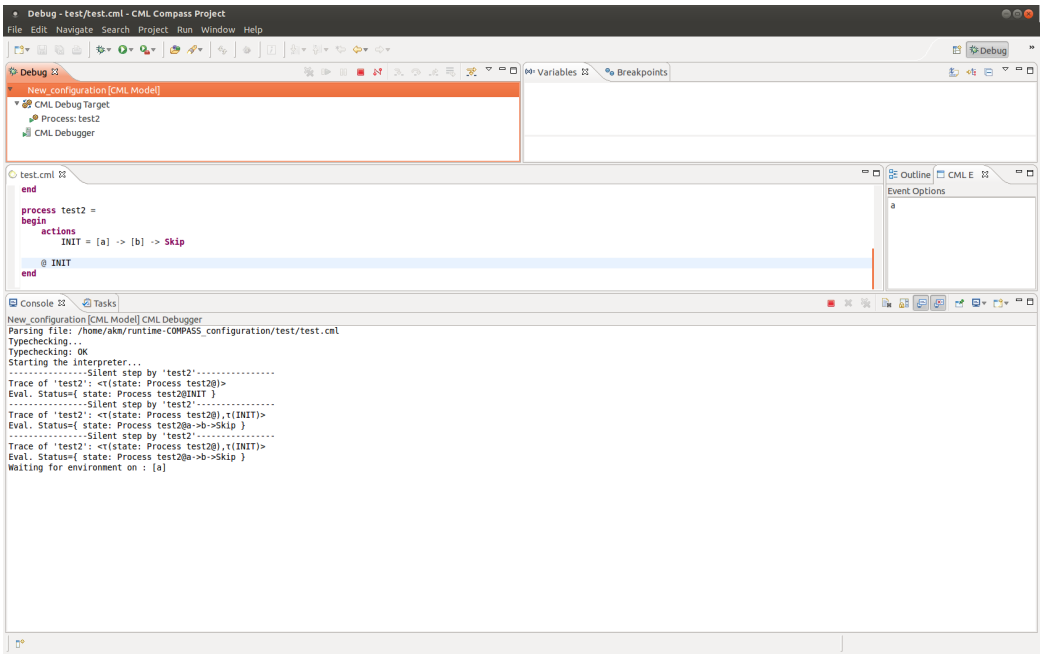

Figure 23: The process “test2” has just been simulated with user interaction. The interpreter is currently waiting for the user to choose an event. In this case the only available event is ’a’

9 Conclusion

As of Month 16 in the COMPASS project we now have the core functionality of the COMPASS IDE in place, and the tool is ready for initial use by the project partners focussing on case studies. This document provides an initial guide to the use of the COMPASS IDE and where to find and activate the tool's features.

This document will be migrated into a wiki form, and future versions will be generated from that. This will allow us to have a continuously-updated user manual with the ability to produce snapshots of it at need.

References

[BGW12] Jeremy Bryans, Andy Galloway, and Jim Woodcock. CML definition 1. Technical report, COMPASS Deliverable, D23.2, September 2012.

[Col13] Joey W. Coleman. Second release of the COMPASS tool — tool grammar reference. Technical report, COMPASS Deliverable, D31.2c, January 2013.

[FLW12] John Fitzgerald, Peter Gorm Larsen, and Jim Woodcock. Modelling and Analysis Technology for Systems of Systems Engineering: Research Challenges. In *INCOSE*, Rome, Italy, July 2012.

[WCF+12] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry. Features of CML: a Formal Modelling Language for Systems of Systems. In *Proceedings of the 7th International Conference on System of System Engineering*, volume 6 of *IEEE Systems Journal*. IEEE, July 2012.